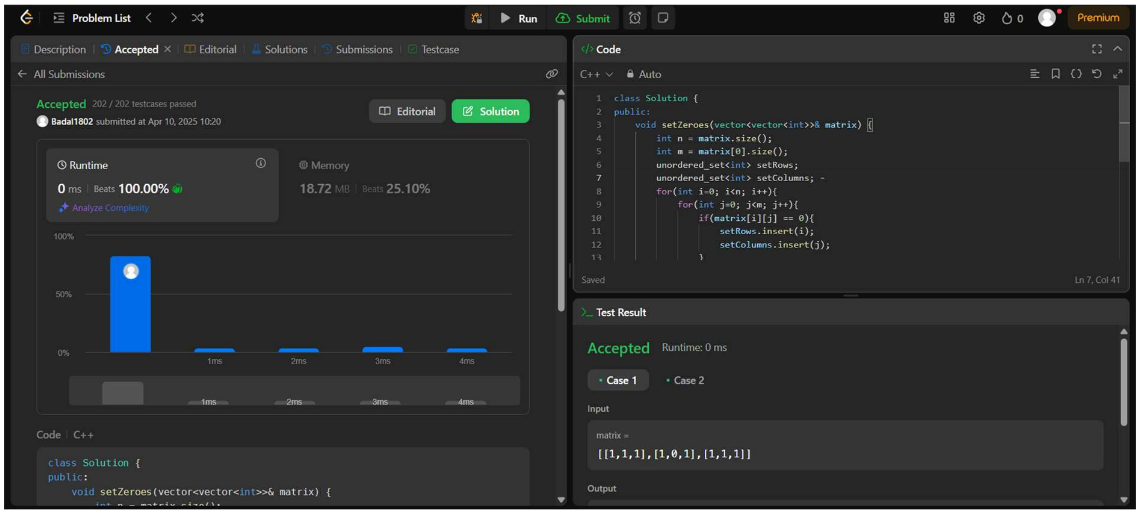
Assignment

1. Set Matrix Zeroes: Given an m x n matrix, if an element is 0, set its entire row and column to 0.

CODE:

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        int n = matrix.size();
        int m = matrix[0].size();
        unordered_set<int> setRows;
        unordered_set<int> setColumns;
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                if(matrix[i][j] == 0){
                    setRows.insert(i);
                    setColumns.insert(j);
                }
            }
        }
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                if(setRows.count(i) > 0 || setColumns.count(j) > 0){
                    matrix[i][j] = 0;
                }
            }
        }
    }
};
```

OUTPUT:



Accepted 202 / 202 testcases passed

Badal1802 submitted at Apr 10, 2025 10:20

Runtime: 0 ms | Beats 100.00%

Memory: 18.72 MB | Beats 25.10%

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        int n = matrix.size();
        int m = matrix[0].size();
        unordered_set<int> setRows;
        unordered_set<int> setColumns;
        for(int i=0; i<n; i++){
            for(int j=0; j<m; j++){
                if(matrix[i][j] == 0){
                    setRows.insert(i);
                    setColumns.insert(j);
                }
```

Accepted Runtime: 0 ms

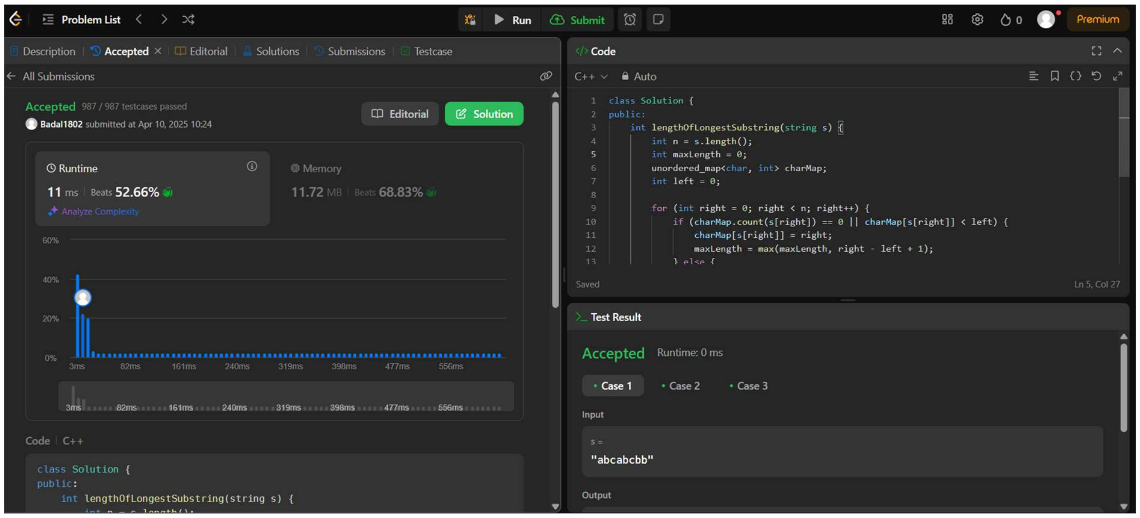
```
matrix =
[[1,1,1],[1,0,1],[1,1,1]]
```

2. Longest Substring Without Repeating Characters: Given a string s, find the length of the longest substring that does not contain any repeating characters.

CODE:

```
class Solution {

public:

    int lengthOfLongestSubstring(string s) {

        int n = s.length();

        int maxLength = 0;

        unordered_map<char, int> charMap;

        int left = 0;


        for (int right = 0; right < n; right++) {

            if (charMap.count(s[right]) == 0 || charMap[s[right]] < left) {

                charMap[s[right]] = right;

                maxLength = max(maxLength, right - left + 1);

            } else {

                left = charMap[s[right]] + 1;

                charMap[s[right]] = right;
            }
        }

        return maxLength;
    }
};
```

OUTPUT:



Accepted 987 / 987 testcases passed
Badal1802 submitted at Apr 10, 2025 10:24

Runtime
11 ms | Beats 52.66%
Analyze Complexity

Memory
11.72 MB | Beats 68.83%

60%
40%
20%
0%
3ms 82ms 161ms 240ms 319ms 398ms 477ms 556ms

Code | C++

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
```

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int n = s.length();
        int maxLength = 0;
        unordered_map<char, int> charMap;
        int left = 0;

        for (int right = 0; right < n; right++) {
            if (charMap.count(s[right]) == 0 || charMap[s[right]] < left) {
                charMap[s[right]] = right;
                maxLength = max(maxLength, right - left + 1);
            } else {
```

Saved
Ln 5, Col 27

Accepted Runtime: 0 ms

Input

```
s =
"abcabcbb"
```

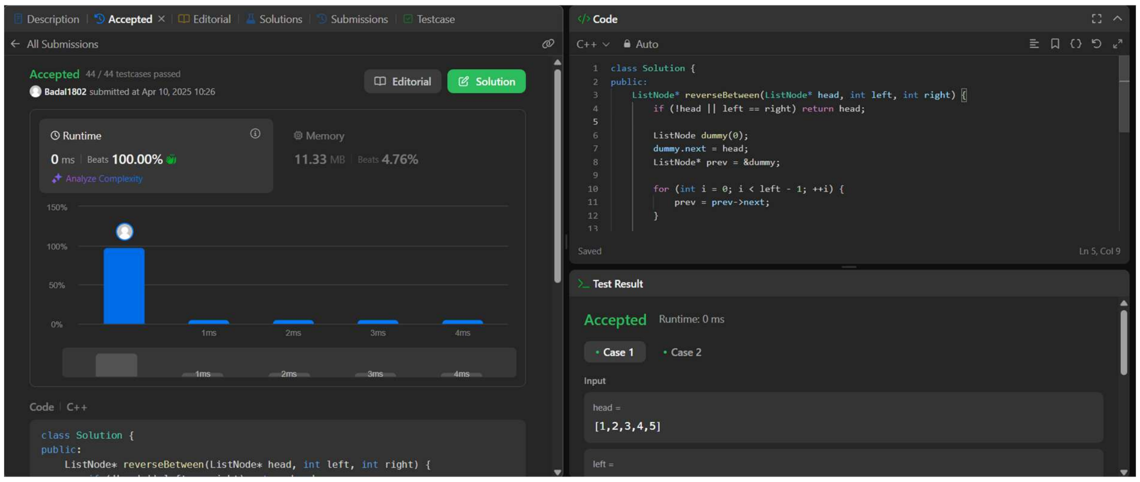
Output

3.Reverse Linked List II: Given the head of a singly linked list and two integers left and right, reverse the nodes of the list from position left to right.

CODE:

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        if (!head || left == right) return head;

        ListNode dummy(0);
        dummy.next = head;
        ListNode* prev = &dummy;
        for (int i = 0; i < left - 1; ++i) {
            prev = prev->next;
        }

        ListNode* current = prev->next;

        for (int i = 0; i < right - left; ++i) {
            ListNode* next_node = current->next;
            current->next = next_node->next;
            next_node->next = prev->next;
            prev->next = next_node;
        }

        return dummy.next;
    }
};
```

OUTPUT:



Accepted 44 / 44 testcases passed
Badal1802 submitted at Apr 10, 2025 10:26

Runtime
0 ms | Beats 100.00%

Memory
11.33 MB | Beats 4.76%

Code | C++

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
```

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        if (!head || left == right) return head;

        ListNode dummy(0);
        dummy.next = head;
        ListNode* prev = &dummy;

        for (int i = 0; i < left - 1; ++i) {
            prev = prev->next;
        }
```

Accepted Runtime: 0 ms

Input

head =
[1,2,3,4,5]

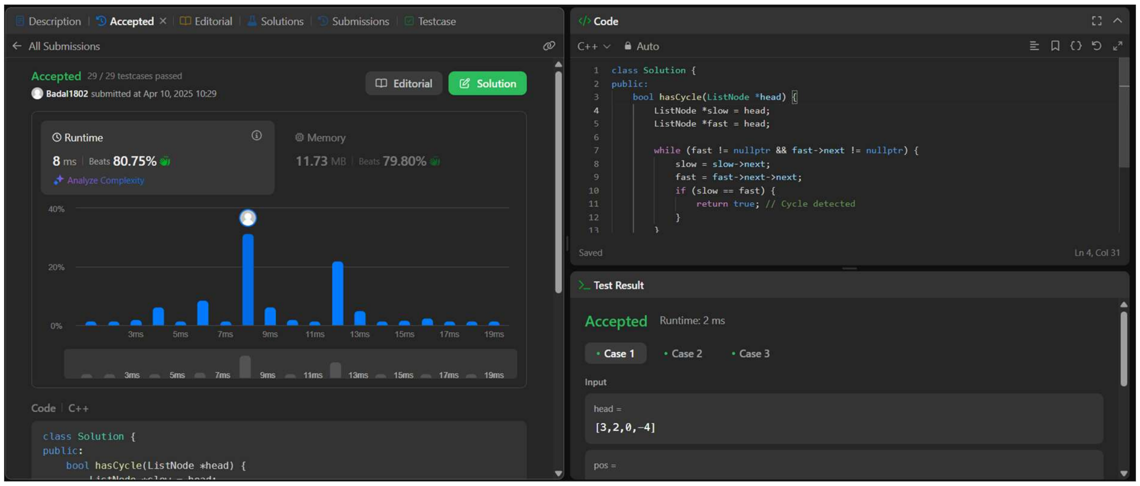
left =

4.Detect a Cycle in a Linked List: Given the head of a linked list, determine whether the linked list contains a cycle. A cycle occurs if a node's next pointer points to a previous node in the list.

CODE:

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode *slow = head;
        ListNode *fast = head;
        while (fast != nullptr && fast->next != nullptr) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) {
                return true; // Cycle detected
            }
        }
        return false; // No cycle
    }
};
```

OUTPUT:



Accepted 29 / 29 testcases passed
Badal1802 submitted at Apr 10, 2025 10:29

Runtime
8 ms | Beats 80.75%
Analyze Complexity

Memory
11.73 MB | Beats 79.80%

Code | C++

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
```

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        ListNode *slow = head;
        ListNode *fast = head;

        while (fast != nullptr && fast->next != nullptr) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) {
                return true; // Cycle detected
            }
        }
```

Saved   Ln 4, Col 31

Accepted Runtime: 2 ms

Case 1 · Case 2 · Case 3

Input

head =
[3,2,0,-4]

pos =

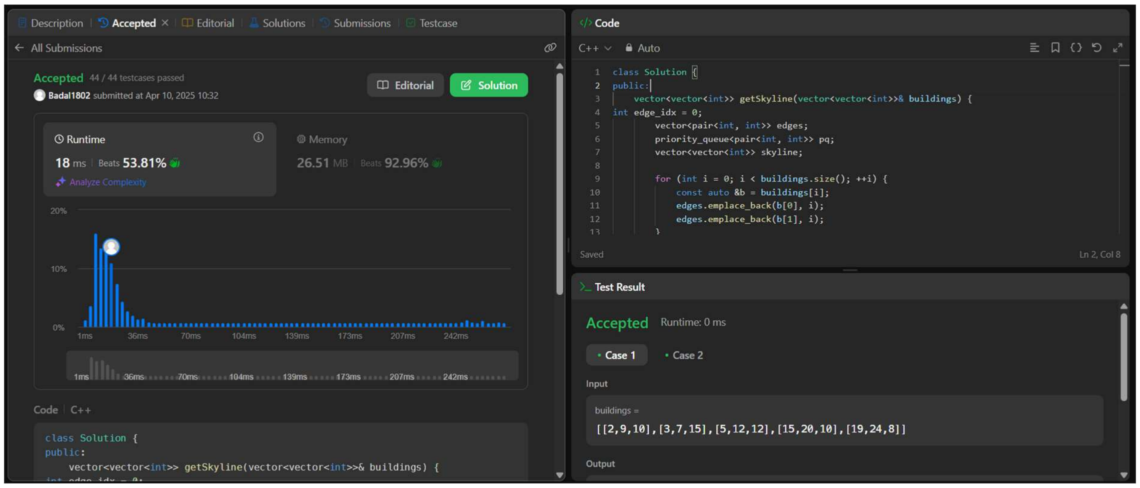
5.The Skyline Problem: Given a list of buildings represented as [left, right, height], where each building is a rectangle, return the key points of the skyline. A key point is represented as [x, y], where x is the x coordinate where the height changes to y

CODE:

```
class Solution {
public:
   vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
int edge_idx = 0;
       vector<pair<int, int>> edges;
       priority_queue<pair<int, int>> pq;
       vector<vector<int>> skyline;

       for (int i = 0; i < buildings.size(); ++i) {
          const auto &b = buildings[i];
          edges.emplace_back(b[0], i);
          edges.emplace_back(b[1], i);
       }

       std::sort(edges.begin(), edges.end());

       while (edge_idx < edges.size()) {
          int curr_height;
          const auto &[curr_x, _] = edges[edge_idx];
          while (edge_idx < edges.size() &&
                 curr_x == edges[edge_idx].first) {
             const auto &[_, building_idx] = edges[edge_idx];
             const auto &b = buildings[building_idx];
             if (b[0] == curr_x)
                pq.emplace(b[2], b[1]);
             ++edge_idx;
```

```
            }
            while (!pq.empty() && pq.top().second <= curr_x)
                pq.pop();
            curr_height = pq.empty() ? 0 : pq.top().first;
            if (skyline.empty() || skyline.back()[1] != curr_height)
                skyline.push_back({curr_x, curr_height});
        }
        return skyline;
    }
};
```

OUTPUT:



Accepted 44 / 44 testcases passed
Badal1802 submitted at Apr 10, 2025 10:32

Runtime 18 ms | Beats 53.81%
Memory 26.51 MB | Beats 92.96%

Code | C++

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
```

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
int edge_idx = 0;
        vector<pair<int, int>> edges;
        priority_queue<pair<int, int>> pq;
        vector<vector<int>> skyline;

        for (int i = 0; i < buildings.size(); ++i) {
            const auto &b = buildings[i];
            edges.emplace_back(b[0], i);
            edges.emplace_back(b[1], i);
        }
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

buildings =
[[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]

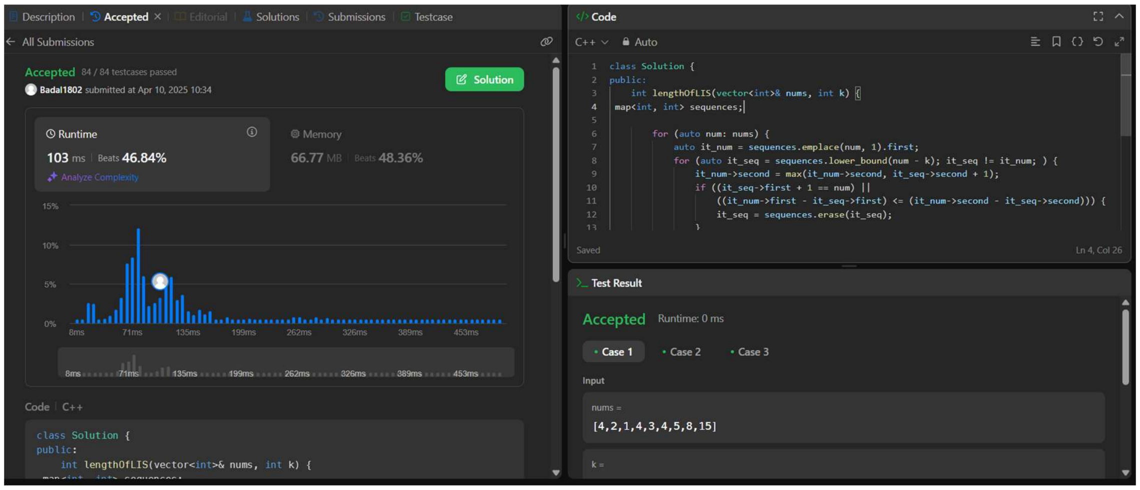
Output

6.Longest Increasing Subsequence II: Given an integer array nums, find the length of the longest strictly increasing subsequence. A subsequence is derived from the array by deleting some or no elements without changing the order of the remaining elements.

CODE:

```
class Solution {
public:
   int lengthOfLIS(vector<int>& nums, int k) {
 map<int, int> sequences;
        for (auto num: nums) {
            auto it_num = sequences.emplace(num, 1).first;
            for (auto it_seq = sequences.lower_bound(num - k); it_seq != it_num; ) {
                it_num->second = max(it_num->second, it_seq->second + 1);
                if ((it_seq->first + 1 == num) ||
                    ((it_num->first - it_seq->first) <= (it_num->second - it_seq->second))) {
                    it_seq = sequences.erase(it_seq);
                }
                else {
                    ++it_seq;
                }
            }
        }
        return max_element(sequences.begin(), sequences.end(), [](auto s1, auto s2) { return s1.second < s2.second; })->second;
    }
};
```

OUTPUT:



Accepted 84 / 84 testcases passed
Badal1802 submitted at Apr 10, 2025 10:34

Runtime
103 ms | Beats 46.84%

Memory
66.77 MB | Beats 48.36%

Code | C++

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums, int k) {
```

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums, int k) {
 map<int, int> sequences;

        for (auto num: nums) {
            auto it_num = sequences.emplace(num, 1).first;
            for (auto it_seq = sequences.lower_bound(num - k); it_seq != it_num; ) {
                it_num->second = max(it_num->second, it_seq->second + 1);
                if ((it_seq->first + 1 == num) ||
                    ((it_num->first - it_seq->first) <= (it_num->second - it_seq->second))) {
                    it_seq = sequences.erase(it_seq);
                }
```

Ln 4, Col 26

Accepted Runtime: 0 ms

Input

nums =
[4,2,1,4,3,4,5,8,15]

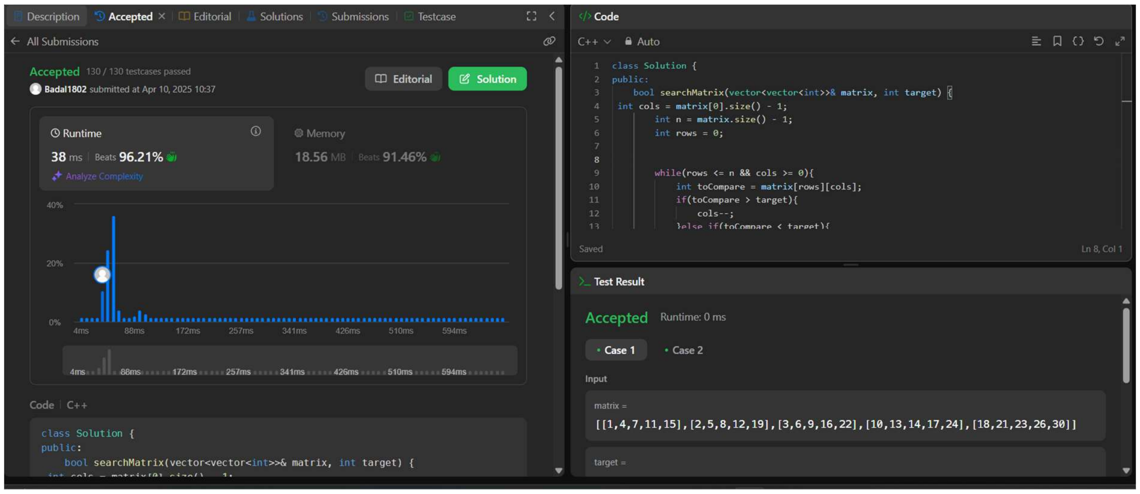
k =

7.Search a 2D Matrix II: Given an m x n matrix where each row is sorted in ascending order from left to right and each column is sorted in ascending order from top to bottom, and an integer target, determine if the target exists in the matrix.

CODE:

```
class Solution {
public:

    bool searchMatrix(vector<vector<int>>& matrix, int target) {
 int cols = matrix[0].size() - 1;
        int n = matrix.size() - 1;
        int rows = 0;

        while(rows <= n && cols >= 0){
            int toCompare = matrix[rows][cols];

            if(toCompare > target){
                cols--;

            }else if(toCompare < target){
                rows++;
            }else{
                return true;
            }

        }
        return false;
    }

};
```

OUTPUT:



Accepted 130 / 130 testcases passed

Badal1802 submitted at Apr 10, 2025 10:37

Runtime 38 ms | Beats 96.21%

Memory 18.56 MB | Beats 91.46%

Code | C++

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
```

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
 int cols = matrix[0].size() - 1;
        int n = matrix.size() - 1;
        int rows = 0;


        while(rows <= n && cols >= 0){
            int toCompare = matrix[rows][cols];
            if(toCompare > target){
                cols--;
            }else if(toCompare < target){
```

Saved Ln 8, Col 1

Accepted Runtime: 0 ms

matrix =
[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]

target =

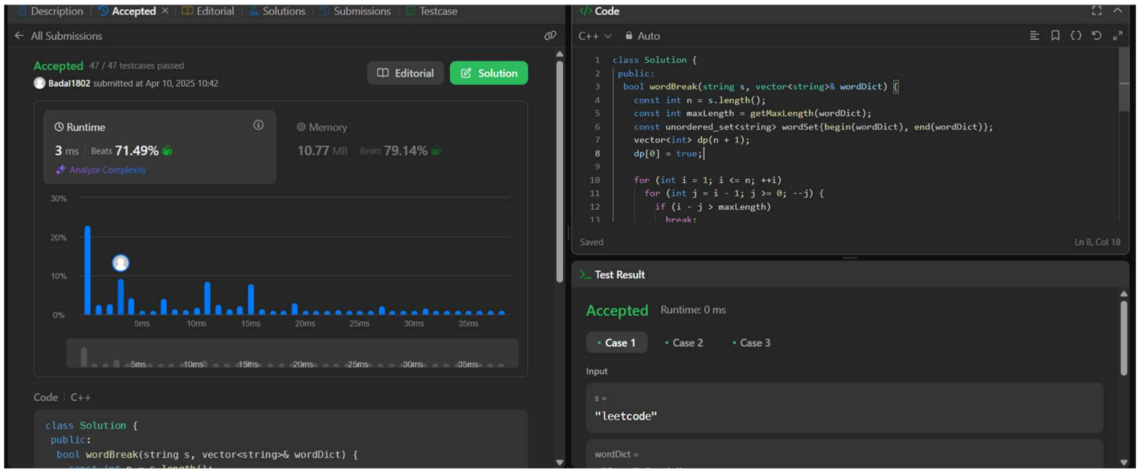
8.Word Break: Given a string s and a dictionary wordDict containing a list of words, determine if s can be segmented into a space-separated sequence of one or more dictionary words. The same word can be reused multiple times.

CODE:

```
class Solution {
 public:
  bool wordBreak(string s, vector<string>& wordDict) {
    const int n = s.length();
    const int maxLength = getMaxLength(wordDict);
    const unordered_set<string> wordSet{begin(wordDict), end(wordDict)};
    vector<int> dp(n + 1);
    dp[0] = true;

    for (int i = 1; i <= n; ++i)
      for (int j = i - 1; j >= 0; --j) {
        if (i - j > maxLength)
```

```
          break;
        if (dp[j] && wordSet.count(s.substr(j, i - j))) {
          dp[i] = true;
          break;
        }
      }

    return dp[n];
  }

 private:
  int getMaxLength(const vector<string>& wordDict) {
    return max_element(begin(wordDict), end(wordDict),
                       [](const auto& a, const auto& b) {
                         return a.length() < b.length();
                       })
        ->length();
  }
};
```

OUTPUT:



Accepted 47 / 47 testcases passed
Badal1802 submitted at Apr 10, 2025 10:42

Runtime: 3 ms | Beats 71.49%
Memory: 10.77 MB | Beats 79.14%

```
class Solution {
 public:
  bool wordBreak(string s, vector<string>& wordDict) {
```

```
class Solution {
 public:
  bool wordBreak(string s, vector<string>& wordDict) {
    const int n = s.length();
    const int maxLength = getMaxLength(wordDict);
    const unordered_set<string> wordSet{begin(wordDict), end(wordDict)};
    vector<int> dp(n + 1);
    dp[0] = true;

    for (int i = 1; i <= n; ++i)
      for (int j = i - 1; j >= 0; --j) {
        if (i - j > maxLength)
          break;
```

Test Result
Accepted Runtime: 0 ms
Case 1 · Case 2 · Case 3
Input
s =
"leetcode"
wordDict =

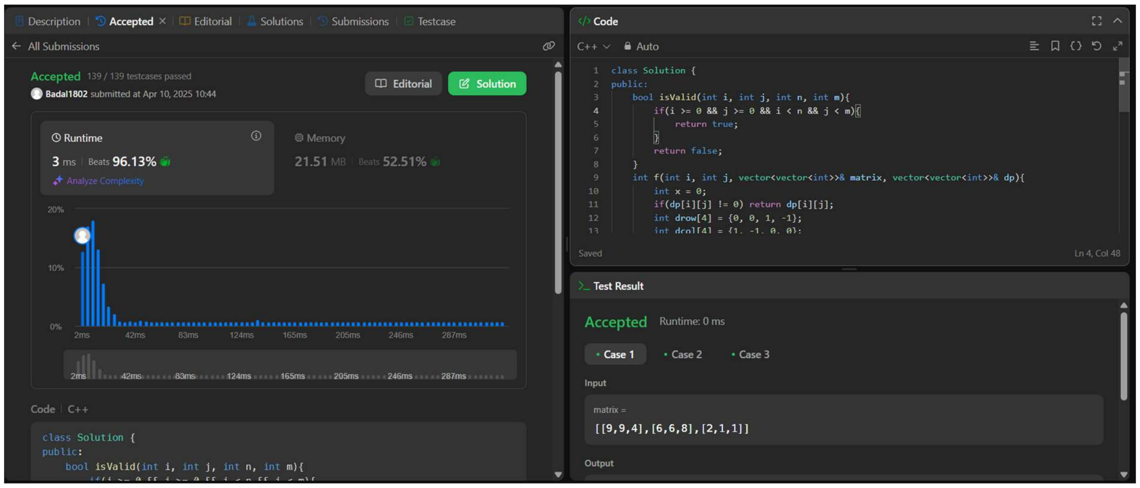
9.Longest Increasing Path in a Matrix: Given an m x n integer matrix, find the length of the longest strictly increasing path. You can move up, down, left, or right from each cell. Diagonal moves and moves outside the boundaries are not allowed.

CODE:

```
class Solution {
public:
  bool isValid(int i, int j, int n, int m){
    if(i >= 0 && j >= 0 && i < n && j < m){
      return true;
    }
    return false;
  }
  int f(int i, int j, vector<vector<int>>& matrix, vector<vector<int>>& dp){
    int x = 0;
    if(dp[i][j] != 0) return dp[i][j];
    int drow[4] = {0, 0, 1, -1};
    int dcol[4] = {1, -1, 0, 0};
    for(int k = 0; k <4; k++){
      int nrow = i+drow[k];
      int ncol = j+dcol[k];
      if(isValid(nrow, ncol, matrix.size(), matrix[0].size()) && matrix[nrow][ncol] >
matrix[i][j]){
        x = max(x, f(nrow, ncol, matrix, dp));
      }
    }
    return dp[i][j] = x+1;
  }
  int longestIncreasingPath(vector<vector<int>>& matrix) {
    int n = matrix.size();
    int m = matrix[0].size();
```

```
        vector<vector<int>>dp(n, vector<int>(m, 0));
        int ans = 0;
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                ans = max(ans, f(i, j, matrix, dp));
            }
        }
        return ans;
    }
};
```

OUTPUT:



Accepted 139 / 139 testcases passed
Badal1802 submitted at Apr 10, 2025 10:44

Runtime: 3 ms | Beats 96.13%
Memory: 21.51 MB | Beats 52.51%

```
class Solution {
public:
    bool isValid(int i, int j, int n, int m){
        if(i >= 0 && j >= 0 && i < n && j < m){
            return true;
        }
        return false;
    }
    int f(int i, int j, vector<vector<int>>& matrix, vector<vector<int>>& dp){
        int x = 0;
        if(dp[i][j] != 0) return dp[i][j];
        int drow[4] = {0, 0, 1, -1};
        int dcol[4] = {1, -1, 0, 0};
```

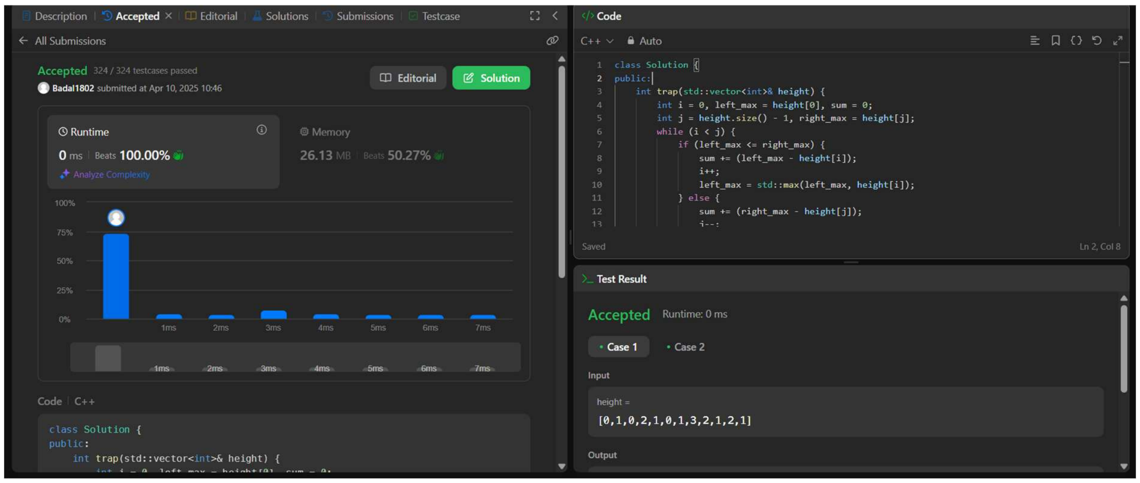
Test Result
Accepted Runtime: 0 ms
Case 1 Case 2 Case 3
Input
matrix =
```
[[9,9,4],[6,6,8],[2,1,1]]
```
Output

10.Trapping Rain Water: Given n non-negative integers representing an elevation map where the width of each bar is 1, compute the total amount of water that can be trapped after raining.

CODE:

```
class Solution {
public:
    int trap(std::vector<int>& height) {
        int i = 0, left_max = height[0], sum = 0;
        int j = height.size() - 1, right_max = height[j];
        while (i < j) {
            if (left_max <= right_max) {
                sum += (left_max - height[i]);
                i++;
                left_max = std::max(left_max, height[i]);
            } else {
                sum += (right_max - height[j]);
                j--;
                right_max = std::max(right_max, height[j]);
            }
        }
        return sum;
    }
};
```

OUTPUT:



Accepted 324 / 324 testcases passed
Badal1802 submitted at Apr 10, 2025 10:46

Runtime 0 ms | Beats 100.00%

Memory 26.13 MB | Beats 50.27%

```
class Solution {
public:
    int trap(std::vector<int>& height) {
```

```
class Solution {
public:
    int trap(std::vector<int>& height) {
        int i = 0, left_max = height[0], sum = 0;
        int j = height.size() - 1, right_max = height[j];
        while (i < j) {
            if (left_max <= right_max) {
                sum += (left_max - height[i]);
                i++;
                left_max = std::max(left_max, height[i]);
            } else {
                sum += (right_max - height[j]);
```

Accepted Runtime: 0 ms

Input

height =
[0,1,0,2,1,0,1,3,2,1,2,1]

Output